(B)[7 Marks] Write a recursive private member function called **`sumAboveItem`** to be included in class `binaryTreeType`. The function returns the sum of the `info` of all nodes in a binary tree whose `info` is larger than `item`. The variable `item` is passed as parameter. Assume that the nodes of the binary tree contain numbers as the `info`.

This function is called from a public member function `treesumAboveItem`, given as follows:

```
template<class Type>
Type binaryTreeType<Type>::treesumAboveItem(Type& item)
{
     return  sumAboveItem(root, item);
}
```

Function Prototype: binary treeType<Type>::

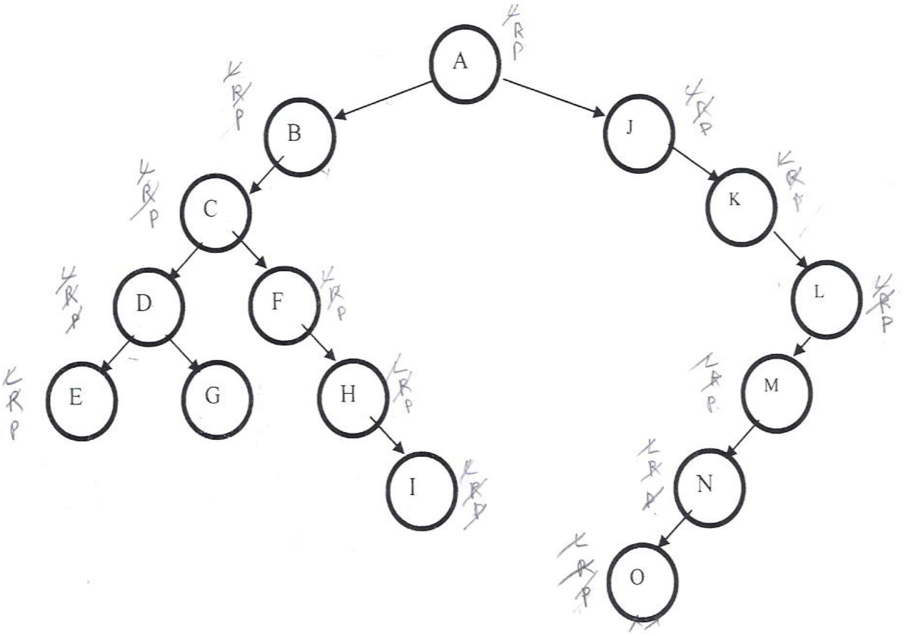
```
     Type sumAboveItem(nodeType<Type>    *p, Type&    item);
   {
        if(p==NULL)
          return Ø;                                  ✓
        else
        {  if(p→info>item)
          return p→info +sumAboveitem(p→Llink, item) +sumAboveItem(p→rlink, item);
          else
          return 0 + sumAboveitem(p→Llink, item)+sumAboveitem(p→rlink, item);

        }

   }
```

6.5

14.5

Question 3 [8 + 7 Marks]

(A) For the binary tree given below, answer the following questions:



i. [1 Marks] What is the **level** of node having info H in this binary tree?

1 level = 4 ✓

ii. [2 Marks] List the **leaf nodes** of this binary tree.

2 E, G, I, O ✓

iii. [5 Marks] List the sequence of nodes, if the binary tree is traversed using **post-order traversal**.

LRP

E, G, D, I, H, F, C, B, O, N, M, L, K, J, A

✓

5

Question 2 [10 Marks]

Write a non-member function **createQueue** that accepts two objects **q1** and **q2** of type **queueType** as parameters. The function is also having a third parameter **item** of type **Type**. If the summation of the first two elements of **q1** is greater than **item** then the function inserts the summation in **q2**, else it inserts **item** in **q2**. The same process will be repeated for the third and fourth elements of **q1** and so on. If the number of elements in **q1** is odd then ignore the last element of **q1** for creating **q2**. If **q1** is empty return false, otherwise return true.

Function prototype:

bool createQueue(queueType<Type>& q1, queueType<Type>& q2, Type& item);

Example:

item = 60

	front						rear	
q1	30	60	13	33	20	43	30	

	front	60	rear					
q2	90	80	63					

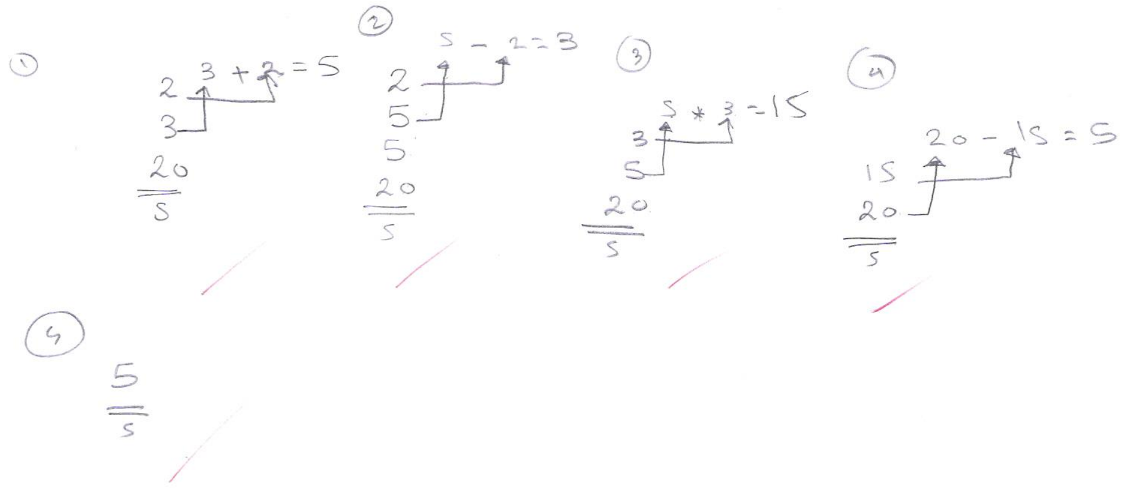
Assume that class **queueType** is available for use. Use only common queue operations such as addQueue, deleteQueue, front, back, isEmptyQueue, isFullQueue, operator= and copy constructor

createQueue(queueType<Type>&q1, queueType<Type>&q2, Type& item)

6

(B) [6 Marks] Consider the following postfix expression. Use stack to evaluate it and show all the push and pop operations by clearly drawing the stack status.

20 3 2 + 5 2 – * –



Question 1 [9 + 6 Marks]

(A) [9 Marks] Write a non-member function called **compareStacks** that accepts two objects ***st1*** and ***st2*** of type **stackType**, as parameters. The function compares the elements of the stack ***st1*** and ***st2*** and performs the following actions:

(i) If both the stacks ***st1*** and ***st2*** contain the same elements in the same order then the function returns 0. found1 = True;

(ii) If both the stacks ***st1*** and ***st2*** contain the same elements but in the reverse order then the function returns 1.

(iii) In all other cases, the function returns -1.

Assume that class **stackType** is available for use. Use only common stack operations such as push, pop, top, isEmptyStack, isFullStack, operator= and copy constructor.

Function prototype:

int ~~void~~ `compareStacks(stackType<Type>  &st1, stackType<Type>  &st2);`

9

```
{
    bool found1= true; bool found2= true;
  stakType <Type> S1( st1);
  stackType<Type>s2(st2);
   stackType <Type> S3;

   while(!S1.isEmptystack())
  {
       if( S1 . Top() != s2 .Top())   ✓
        found1= false;
        S1. pop();
        s2 . pop();
  }
   while ( !st2. isemptystack())
  {
        S3 . push( st2.Top())   ✓
        st2. pop();
   }
     while(!S3.isEmpty stack())
   {    if(st1.Top() != S3 . Top())
          found2= false;
          S3 . pop();   ✓
          st1 .pop();
   }
   if ( found1== true)   ✓
     return 0;
   else if( found2 == true)   ✓
     return 1;
   else
     return -1;   ✓
}
```